Searching

CS 240

Alex Vondrak

ajvondrak@csupomona.edu

Winter 2012

Searching

Definition (The Searching Problem)

Given: An array, `E[] haystack` and an item, `E needle`

Return: **`true`** is `needle` can be found at any point in `haystack`; otherwise, **`false`**

I.e., "can we find this element in this sequence?"

Multiple Choice Question

Given the `haystack`

`{3, 1, 4, 1, 5, 9, 2}`

is the needle `1` in the `haystack`?

(A) Yes

(B) No

Multiple Choice Question

What do you suppose our algorithm is for finding `1` in the following array?

```
{3, 1, 4, 1, 5, 9, 2}
```

(A) Look through each element, stop when we see a `1`

(B) Look through each element, stop when we see something that isn't a `1`

(C) Look through every element, but don't stop; note whether we saw `1` anywhere

(D) Look at index `[1]` or `[3]`

Multiple Choice Question

What do you suppose our algorithm is for finding `1` in the following array?

`{8, 6, 7, 5, 3, 0, 9}`

(A) Look through each element, stop when we see a `1`

(B) Look through each element, stop when we see something that isn't a `1`

(C) Look through every element, but don't stop; note whether we saw `1` anywhere

(D) Look at index [1] or [3]

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    ⋮
}
```

How would we code this algorithm?

(A) Use a **`for`**-loop

(B) Use recursion

(C) Either of the above

(D) None of the above

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    for (E item : haystack) {
        ⋮
    }
}
```

What should we do in the body of the **for**-loop?

(A) Check if item is the same thing as the needle

(B) Check if item is not the same as needle

(C) Both of the above

(D) None of the above

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    for (E item : haystack) {
        if (item.equals(needle)) {
            ...
        }
    }
}
```

Why do we use `equals` instead of `==`?

(A) Because == doesn't do a "deep" comparison

(B) Because == only compares addresses

(C) Because `item` and `needle` are objects, not primitive types

(D) All of the above

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    for (E item : haystack) {
        if (item.equals(needle)) {
            ...
        }
    }
}
```

If we've found the `needle`, what should we do?

(A) Set a **`boolean`** flag to **`true`**

(B) Return **`true`**

(C) Increment a counter

(D) Advance to the next element

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    for (E item : haystack) {                                (A)
        if (item.equals(needle)) {                           (B)
            return true;                                     (C)
        }
        else {
            return false;                                    (D)
        }
    }
}
```

Where is the bug in the above code?

Multiple Choice Question

```
public boolean search(E needle, E[] haystack) {
    for (E item : haystack) {
        if (item.equals(needle)) {
            return true;
        }
    }
    return false;
}
```

What is the running time complexity of the above method?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) $O(n^2)$

Multiple Choice Question

Let's make the `search` method recursive.

```
public boolean search(E needle, E[] haystack) {
    if (/* Base Case */) {
        ...
    }
    /* Recursive Case */
}
```

What do you suppose the base case should be?

(A) If `haystack` is empty

(B) If `needle` is equal to `haystack[0]`

(C) If `needle` is **`null`**

(D) None of the above

Multiple Choice Question

Let's make the `search` method recursive.

```
public boolean search(E needle, E[] haystack, int i) {
    if (/* Base Case */) {
        ...
    }
    /* Recursive Case */
}
```

What do you suppose the base case should be *now*?

(A) If `i == 0`

(B) If `needle` is equal to `haystack[0]`

(C) If `needle` is equal to `haystack[i]`

(D) If `i >= haystack.length`

Multiple Choice Question

Let's make the `search` method recursive.

```
public boolean search(E needle, E[] haystack, int i) {
    if (i >= haystack.length) {
        return false;
    }
    /* Recursive Case */
}
```

What is the recursive case?

(A) When `i < haystack.length`

(B) When `needle` is equal to `haystack[i]`

(C) Both of the above

Multiple Choice Question

Let's make the search method recursive.

```
public boolean search(E needle, E[] haystack, int i) {
    if (i >= haystack.length) {
        return false;
    }
    if (needle.equals(haystack[i])) {
        return true;
    }
    /* Recursive Case */
}
```

What is the recursive case?

(A) When i < haystack.length

(B) When needle is not equal to haystack[i]

(C) Both of the above

Multiple Choice Question

Let's make the `search` method recursive.

```
public boolean search(E needle, E[] haystack, int i) {
    if (i >= haystack.length) {
        return false;
    }
    if (needle.equals(haystack[i])) {
        return true;
    }
    return this.search(needle, haystack, i + 1);
}
```

What is the running time complexity of the above method?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) $O(n^2)$

Multiple Choice Question

Let's make the `search` method recursive.

```
public boolean search(E needle, E[] haystack, int i) {
    if (i >= haystack.length) {
        return false;
    }
    if (needle.equals(haystack[i])) {
        return true;
    }
    return this.search(needle, haystack, i + 1);
}
```

Let $T(n)$ be the exact running time of the above method (though, assume constant-time operations only cost 1). How can we express it?

(A) $T(0) = 1$

(B) $T(n) = 1 + T(n-1)$

(C) Both of the above

(D) None of the above

Multiple Choice Question

Can we improve upon the time complexity of the search problem?

(A) Yes
(B) No
(C) Maybe

Binary Search

Definition

Suppose you want to search through a sorted array,

$$S = \{S_1, S_2, S_3, \ldots, S_n\}$$

for a particular element, x.
In general, you're always searching between two indices: L and R.
Initially, it would be between $L = 1$ and $R = n$.

- Let $M = \lfloor (L + R)/2 \rfloor$
- If $L > R$, then x is not in S (Base Case)
- If $S_M = x$, then x is in S (Base Case)
- If $S_M > x$, then search between L and $M - 1$ (Recursive Case)
- If $S_M < x$, then search between $M + 1$ and R (Recursive Case)

Multiple Choice Question

After having coded a binary search recursively, how can we express its running time, $T(n)$?

(A) $T(0) = 1,\ T(n) = 1 + T(n-1)$

(B) $T(1) = 1,\ T(n) = 1 + T(n-1)$

(C) $T(0) = 1,\ T(n) = 1 + T(n/2)$

(D) $T(1) = 1,\ T(n) = 1 + T(n/2)$

Multiple Choice Question

Loosely speaking (to save the headache), we have

$$T(1) = 1$$
$$T(n) = 1 + T(n/2)$$

Proof Sketch ($T(n) = 1 + \log_2 n$).

By induction on n.

Base:

Inductive:

□

What is the base case?

(A) $n = 0$
(B) $n = 1$
(C) $T(n) = 1$
(D) $1 + \log_2(0)$

Multiple Choice Question

Loosely speaking (to save the headache), we have

$$T(1) = 1$$
$$T(n) = 1 + T(n/2)$$

Proof Sketch ($T(n) = 1 + \log_2 n$).

By induction on n.

Base: When $n = 1$, $1 + \log_2(1) = 1 + 0 = 1 \quad = \quad T(n)$

Inductive: As our I.H., assume...

□

What is the Inductive Hypothesis?

(A) $T(1) = 1$

(B) $T(k) = 1 + \log_2(k)$ for some $k \in \mathbb{N}$

(C) $T(r) = 1 + \log_2(r)$ for any $r \leq k \in \mathbb{N}$

(D) $T(k+1) = 1 + \log_2(k)$

Multiple Choice Question

Loosely speaking (to save the headache), we have

$$T(1) = 1$$
$$T(n) = 1 + T(n/2)$$

Proof Sketch ($T(n) = 1 + \log_2 n$).

By induction on n.

Base: When $n = 1$, $1 + \log_2(1) = 1 + 0 = 1 \quad = \quad T(n)$.

Inductive: As our I.H., assume $T(r) = 1 + \log_2(r)$ for any $r \leq k \in \mathbb{N}$.

□

What do we want to be true now?

(A) $T(1) = 1$
(B) $T(k) = 1 + \log_2(k)$ for some $k \in \mathbb{N}$
(C) $T(r) = 1 + \log_2(r)$ for any $r \leq k \in \mathbb{N}$
(D) $T(k+1) = 1 + \log_2(k)$

Loose Proof That Binary Search Is $O(\log_2(n))$

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 1 + T(n/2) \end{aligned}$$

Proof Sketch ($T(n) = 1 + \log_2 n$).

Base: When $n = 1$, $1 + \log_2(1) = 1 + 0 = 1 \quad = \quad T(n)$.

Inductive: As our I.H., assume $T(r) = 1 + \log_2(r)$ for any $r \leq k \in \mathbb{N}$.

$$\begin{aligned} T(k+1) &= 1 + T((k+1)/2) \\ &= 1 + 1 + \log_2((k+1)/2) && \text{(by I.H.)} \\ &= 1 + 1 + \log_2(k+1) - \log_2(2) \\ &= 1 + 1 + \log_2(k+1) - 1 \\ &= 1 + \log_2(k+1) \end{aligned}$$

□

Multiple Choice Question

What's the issue with the Binary Search algorithm?

(A) It will work when the array is unsorted, but slower

(B) It won't work when the array is unsorted

(C) We have to sort the array before a search

(D) Nothing; it's much faster than a linear search

Multiple Choice Question

Can we think of an algorithm better than $O(\log_2(n))$?

(A) Yes

(B) No

(C) Probably